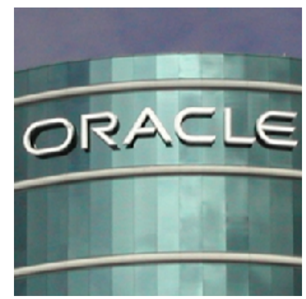
High Performance Data Management - “It's the memory stupid!”

Leveraging system resource characteristics to efficiently improve performance and predictability

Tim Kaldewey [1,2]
Scott Brandt [1]

Andrea di Blas [1,2]
Eric Sedlar [2]



1University of California Santa Cruz

School of Engineering

{kalt, scott, andrea}

@soe.ucsc.edu

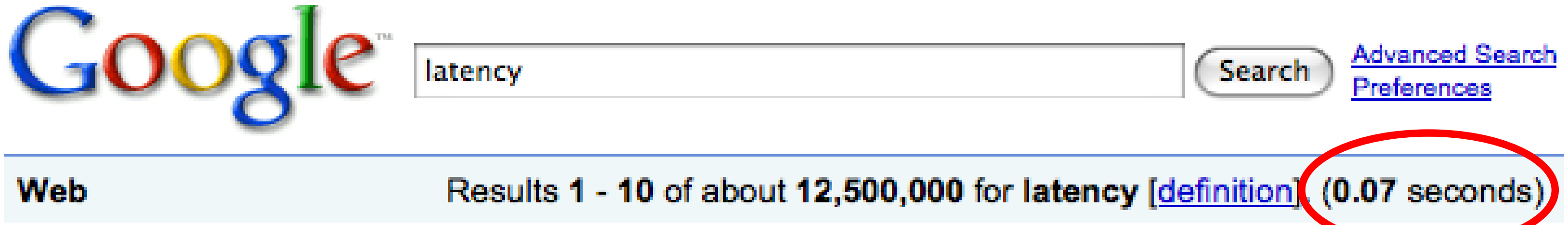
Oracle Corporation

Server Technologies – Special Projects

{tim.kaldewey, andrea.di.blas, eric.sedlar}

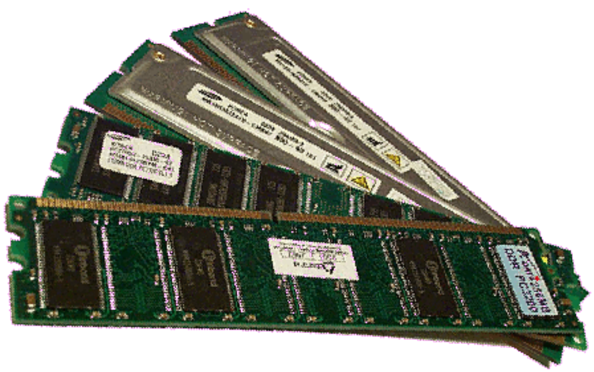
@oracle.com

High Performance Data Management – Bottlenecks



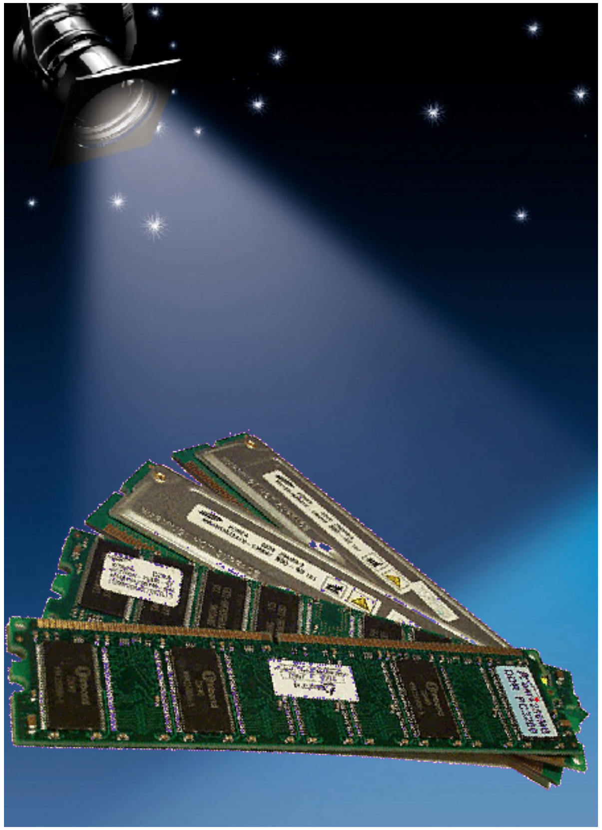
- ~~CPU – GHz~~
- ~~Disk – up to 15ms latency~~
- Memory
  - Performance – 70ns latency
  - Predictability – multi-level caches
  - Rapidly growing sizes

Ping ~40ms
Disk accesses ~15ms



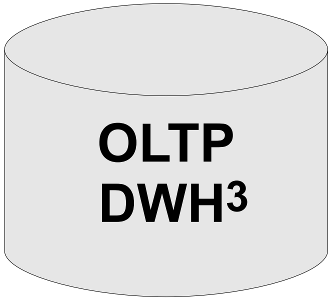
Memory Matters

- Is disk I/O[1] still the bottleneck for traditionally dat intensive applications, e.g. databases[1]?
- “It's the memory Stupid!” [2]



- Growth rates of main memory size have outstripped the growth rates of structured data in the enterprise
- Multiple GB main memory DB put memory performance on the spot

>

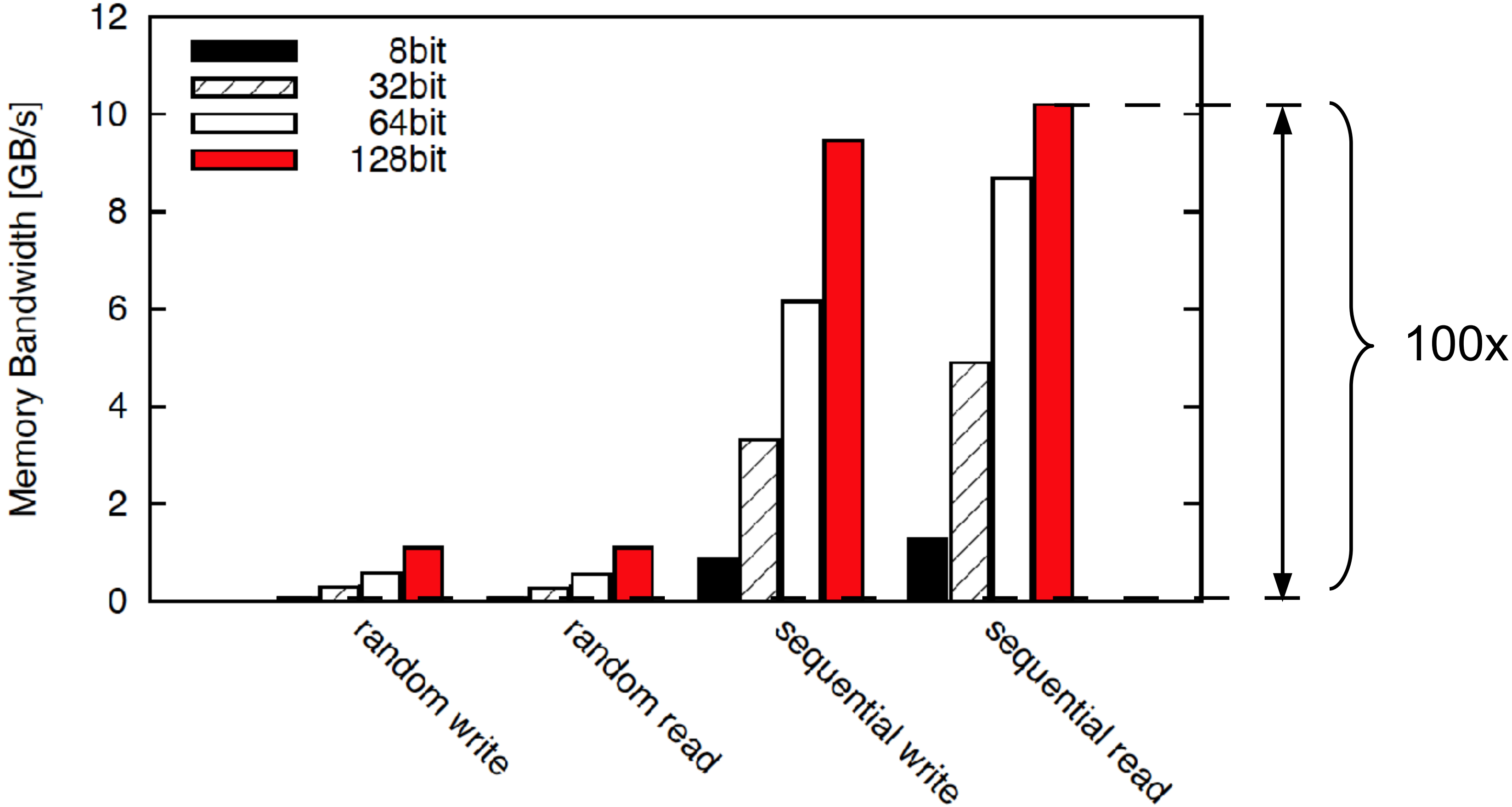


- Isn't memory performance constant ?

[1] A. Ailamaki, et al. DBMSs on a modern processor: Where does time go? VLDB’99
[2] R. Sites. It’s the memory, stupid! MicroprocessorReport, 10(10),1996
[3] K. Schlegel. Emerging Technologies Will Drive Self-Service Business Intelligence. Garter Report 2/08

Memory Performance – Characterization

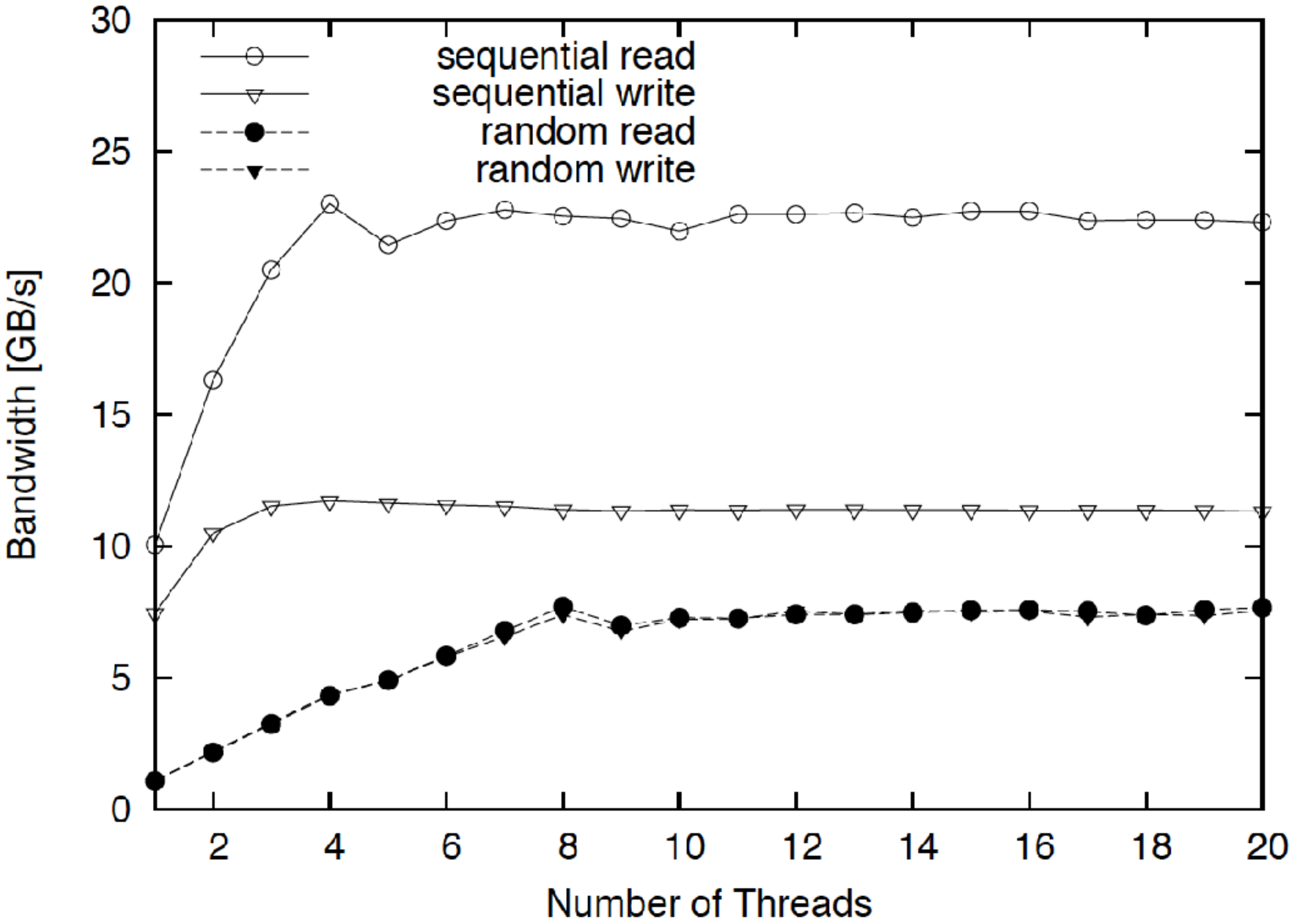
- Dependent on Access pattern and word size performance differs up to 2 orders of magnitude



32GB data accessed total. Results for a Core i7 2.66GHz, DDR3 1666.

Memory Performance – More Characteristics

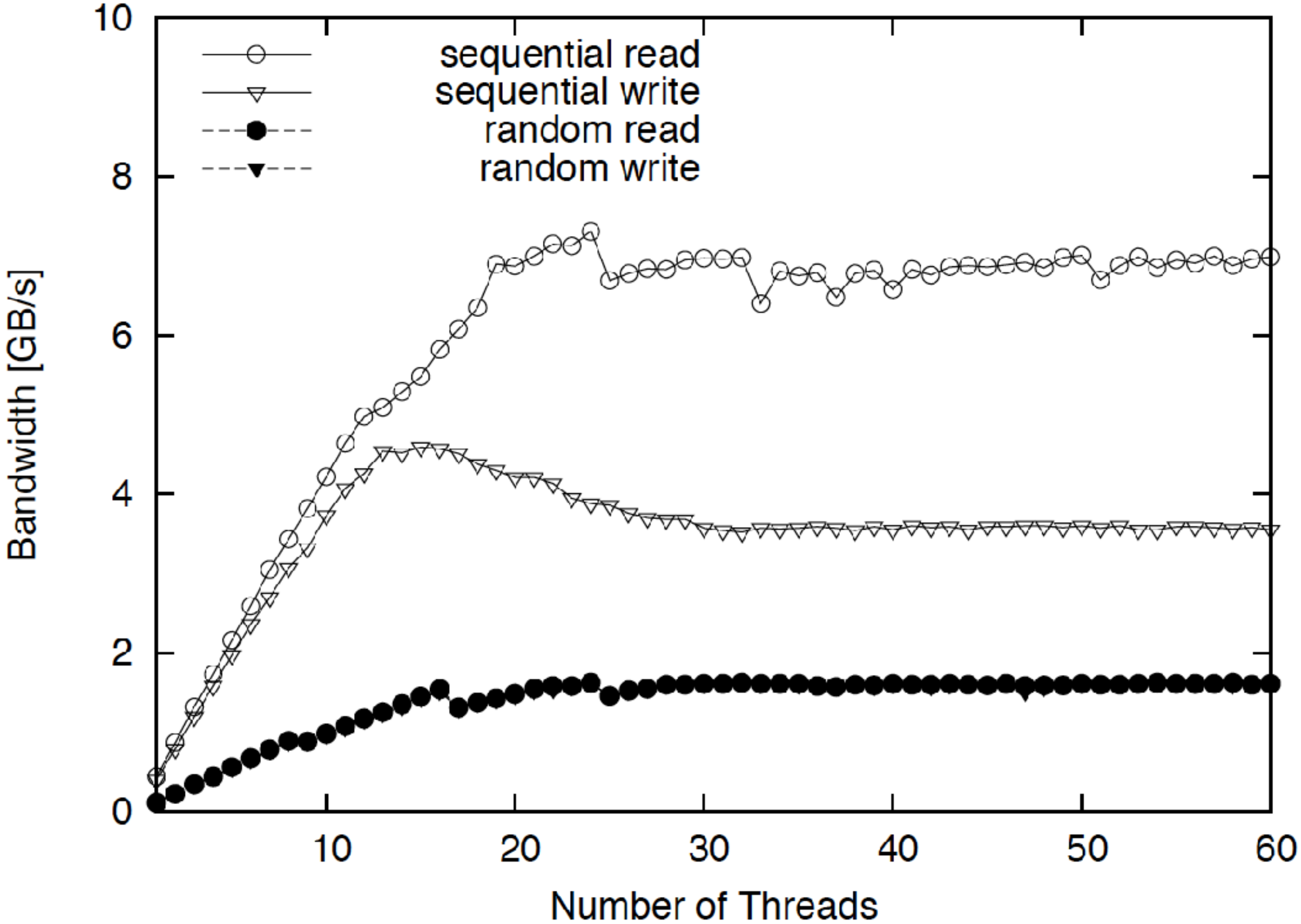
- Peak performance requires parallel memory access



Throughput with increasing number of threads. 32GB of 64-bit words accessed total.
Results for a Core i7 2.66GHz, DDR3 1666.

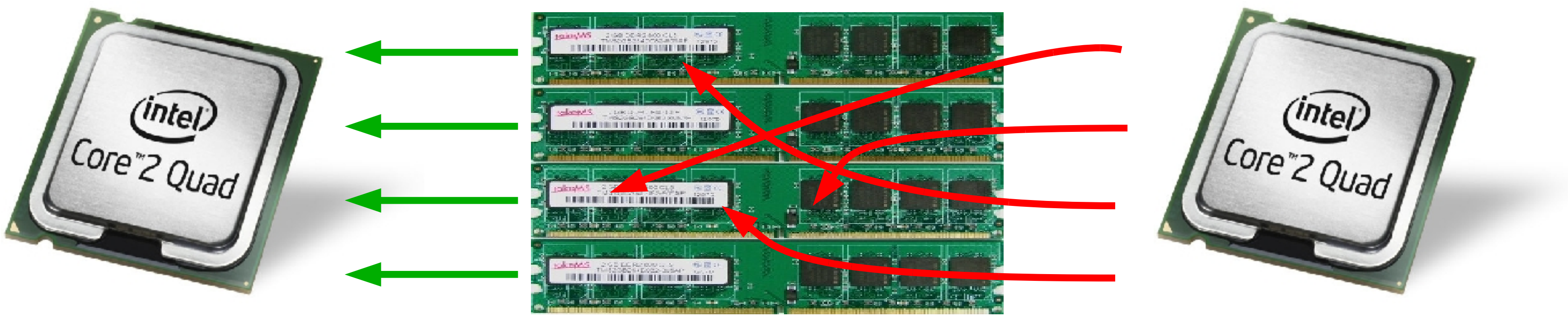
Peak Memory Performance

- Required level of concurrency depends on the architecture



Throughput with increasing number of threads. 32GB of 64-bit words accessed total.
Results for an 8-core Sun Niagara, DDR2 533.

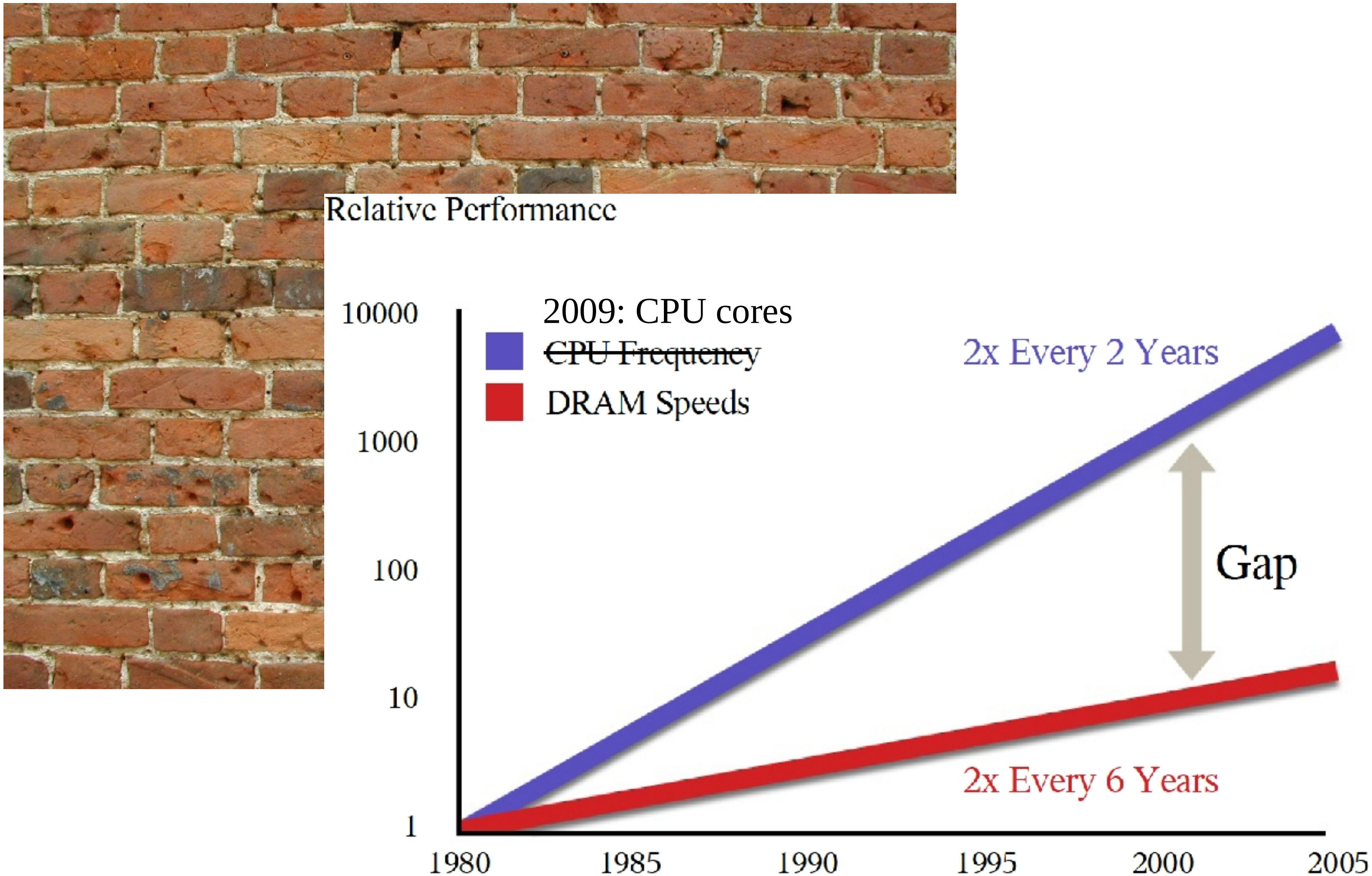
RAM = Random Access memory ?



Aspect	Performance impact	Reason
Access pattern	18x sequential vs. random 2x read vs. write	MTU = Cache Line
Data type	16x 128-bit vs. 8-bit words	Memory Controller
Parallelism	32x multithreaded vs. serial	

What do these results imply?

The (Memory) Wall [4]

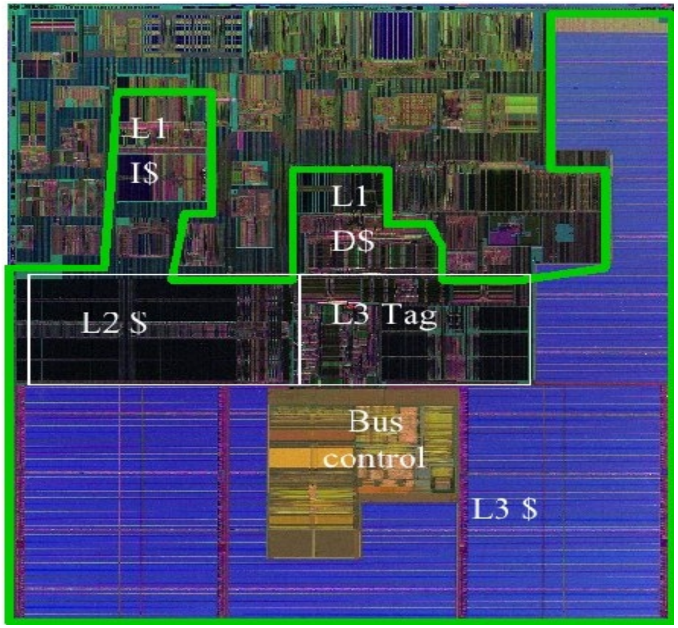


Source: David Yen. Opening Doors to the MultiCore Era. MultiCore Expo 2006

[4] W.A.Wulf et al. Hitting the memory wall: implications of the obvious. SIGARCH - Computer Architecture News'95

Overcoming the Memory Wall – Traditional Approaches

- Larger caches
  - Specialized processors
  - TPC-H top10: 6 run10 Itanium



- “Linearize” data structures
  - For example matrix multiplication: store 1st matrix row-wise, 2nd column-wise (memory is 1D)

1	2	3
4	5	6
7	8	9

X

1	4	7
2	5	8
3	6	9

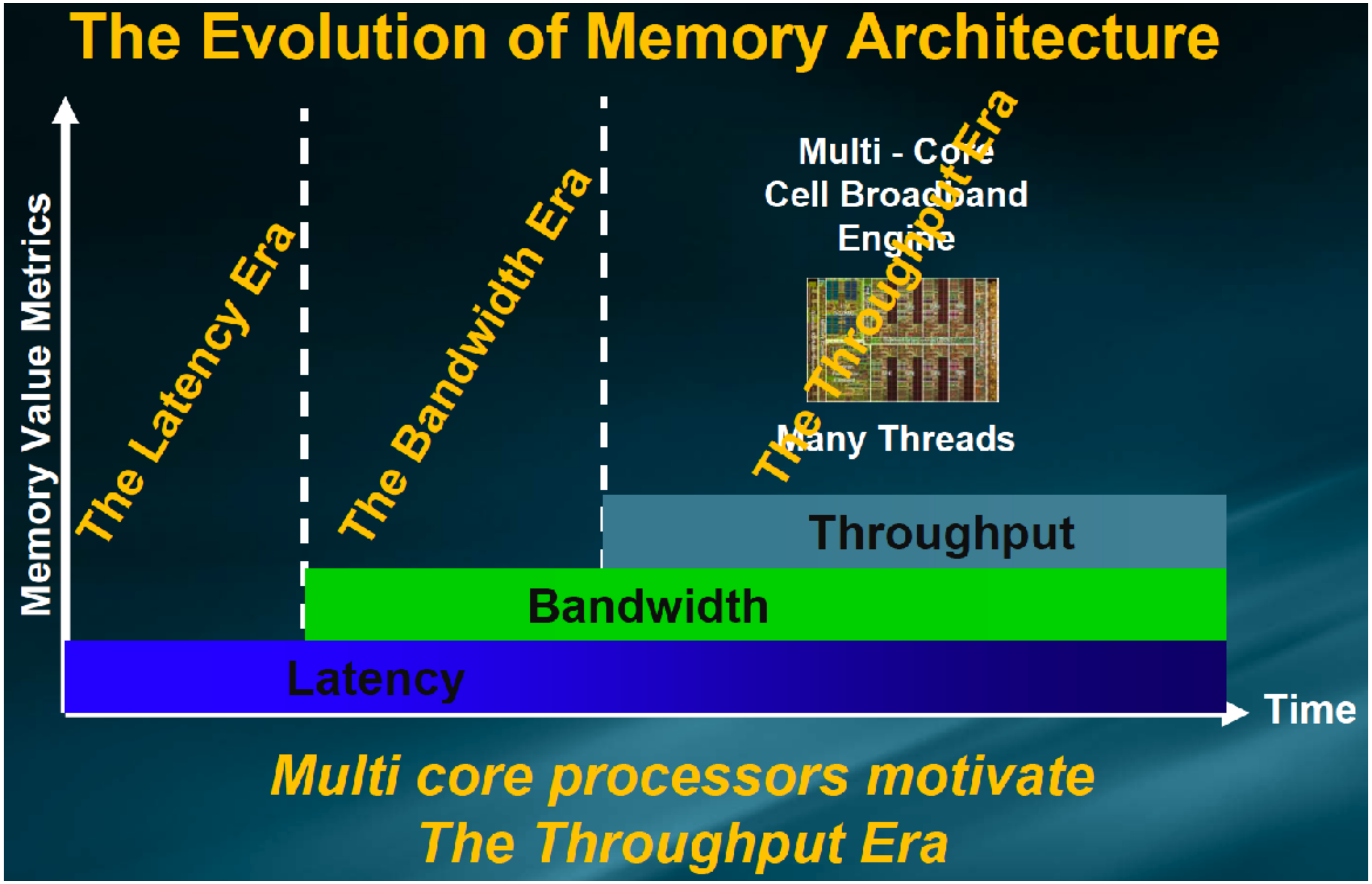
➔

1	2	3	4	5	6	7	8	9

X

1	2	3	4	5	6	7	8	9

Latency & Bandwidth – historical Issues ?

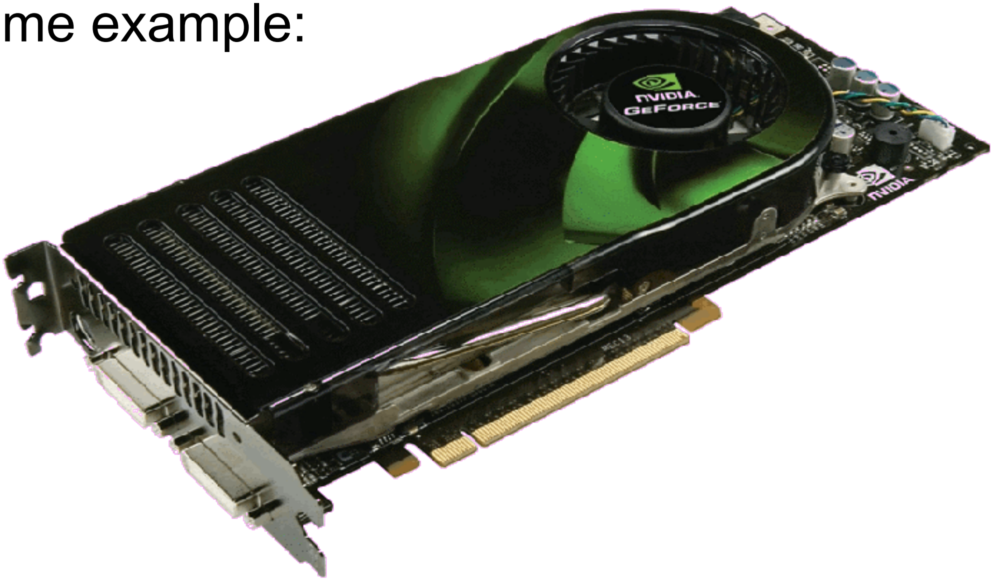


Source: Terabyte Bandwidth Initiative. Craig Hampel - Rambus. HotChips'08

Overcoming the Memory Wall – “Newer” Approaches

- Multithreading
  - Run multiple (similar) jobs simultaneously ➔ increased throughput

an extreme example:



But individual jobs won't get any faster =(

Overcoming the Memory Wall – "Revolutionary" Approaches

- New parallel algorithms
  e.g. p-ary search [5,6]

[5] T. Kaldewey, J. Hagen, A. Di Blas, E.Sedlar. Parallel Search on Video Cards. USENIX Hotchips'09
[6] A. Di Blas, T. Kaldewey. Data Monster. IEEE Spectrum 9/09

Why Search ?

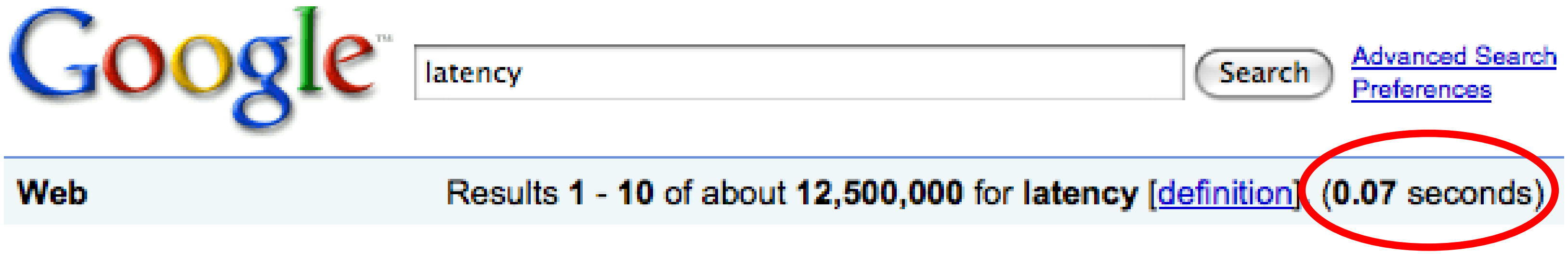
Honestly, how many times a day do you visit

?

Search – A Performance Problem ?

- Large dot-com's server farms handle millions of queries simultaneously
  - High throughput is a “must have”
  - Achieved through (massive) parallelism
- What are we waiting for ?
  - Network latency
  - Response time < sub-second

    ➔ At the data source: query < millisecond(s)



Ping ~40ms
Disk accesses ~15ms
App. overhead ?

Our Goal

- Improve response time (latency) in the era of throughput oriented (parallel) computing.

Research Question ?

- How can we (algorithmically) exploit (memory) parallelism to improve response time (of search)?

Binary Search

- How Do you (efficiently) search an index ?

- Open phone book ~middle
- 1st name = whom you are looking for ?
- < , > ?
- Iterate
  - Each iteration: #entries/2 (n/2)
  - Total time: ➔ $\log_2(n)$

Parallel (Binary) Search

- What if you have some friends (3) to help you ?

- Divide et impera !
  - Each is using binary search takes $\log_2(n/4)$
  - All can work in parallel ➔ faster: $\log_2(n/4) < \log_2(n)$
  - 3 of you are wasting their time !
- Give each of them ¼ *

* You probably want to tear it a little more intelligent than that, e.g. at the binding ;-)

P-ary Search

- Divide et impera !!



…

- How do we know who has the right piece ?

- It's a sorted list:
  - Look at first and last entry of a subset
  - If first entry < searched name < last entry
    - Redistribute
    - Otherwise … throw it away
  - Iterate

P-ary Search

- What do we get

+

- Each iteration: n/4
  ➔ $\log_4(n)$
- Assuming redistribution time is negligible:
  $\log_4(n) < \log_2(n/4) < \log_2(n)$
- But each does 2 lookups !
- How time consuming are lookup and redistribution ?

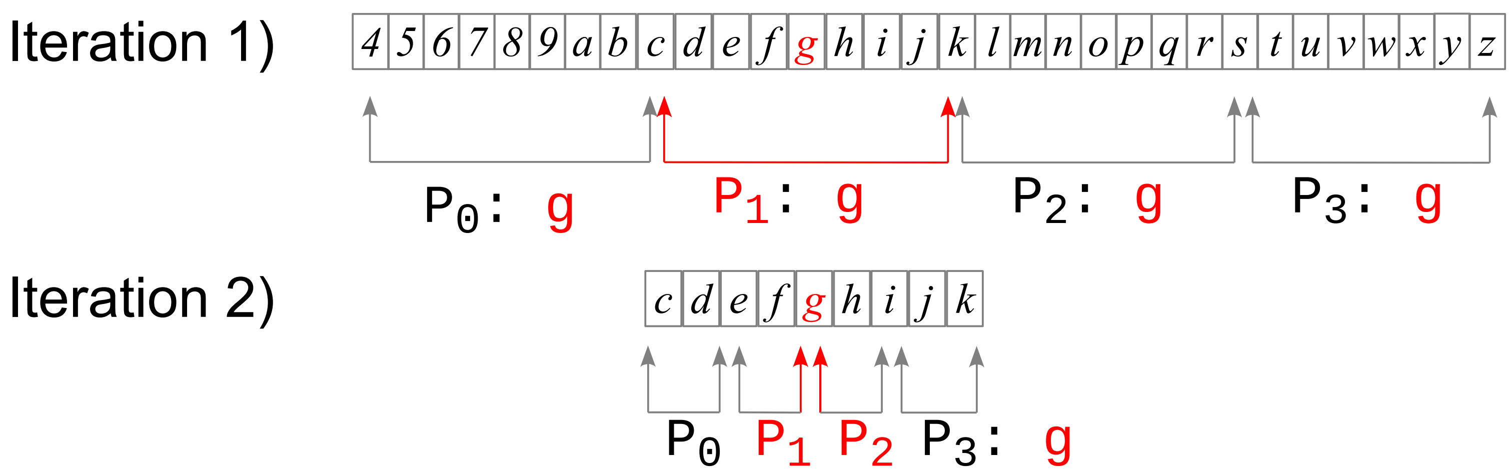
  II II

  memory access synchronization

- Searching a database index can be implemented the same way
  - Without destroying anything ;-)

P-ary search - Implementation

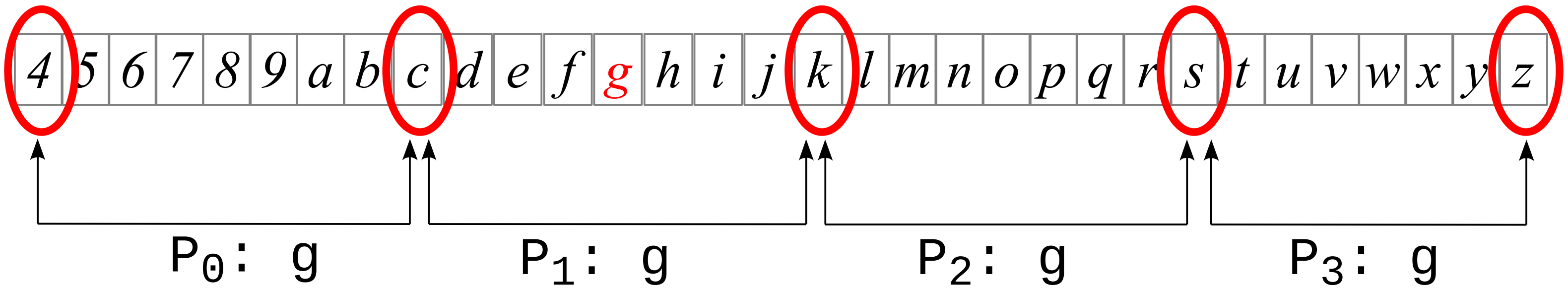
- Performance depends on target architecture
  - # friends = threads / processor cores / vector



  - Redistribution ➔ synchronization cost
    pthreads ($$), spinlock($), SIMD/vector (~0)

P-ary search - Implementation

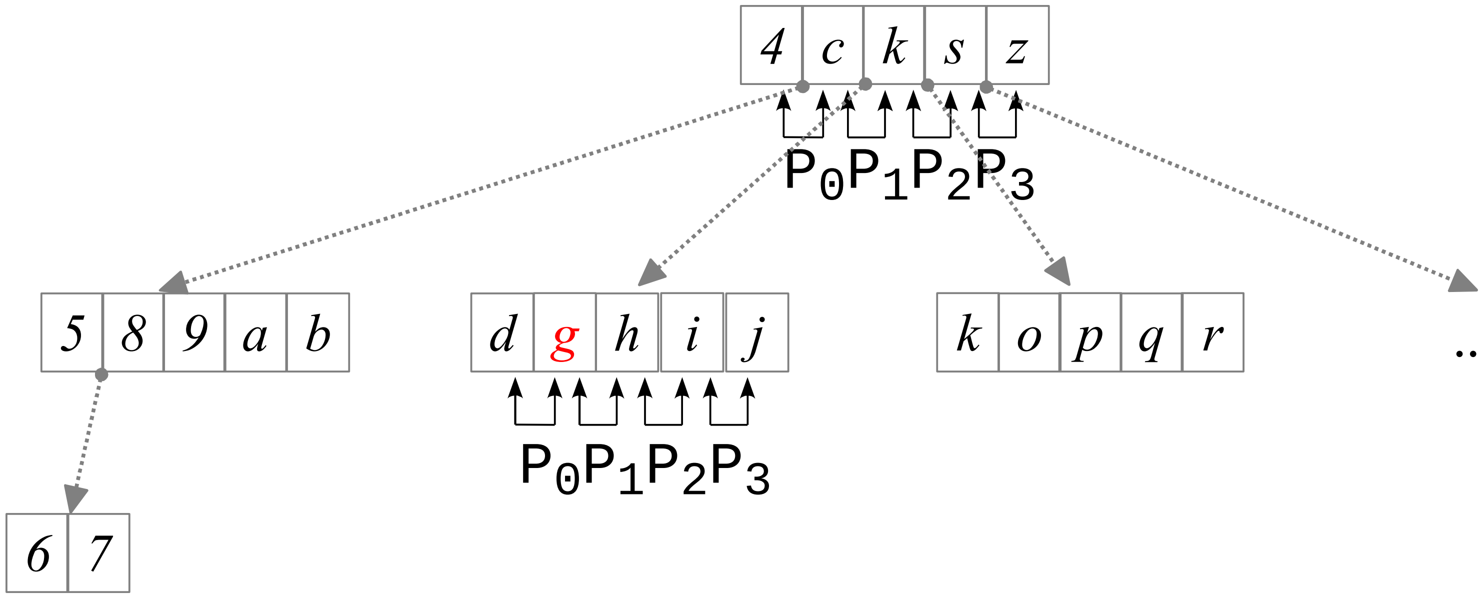
- Performance depends on data structure
  - Sorted lists require multiple lookups *or memory gather*
    ➔ random accesses



  - Random memory accesses are slow
  - Memory gather not (yet) available for vector units (SSE)

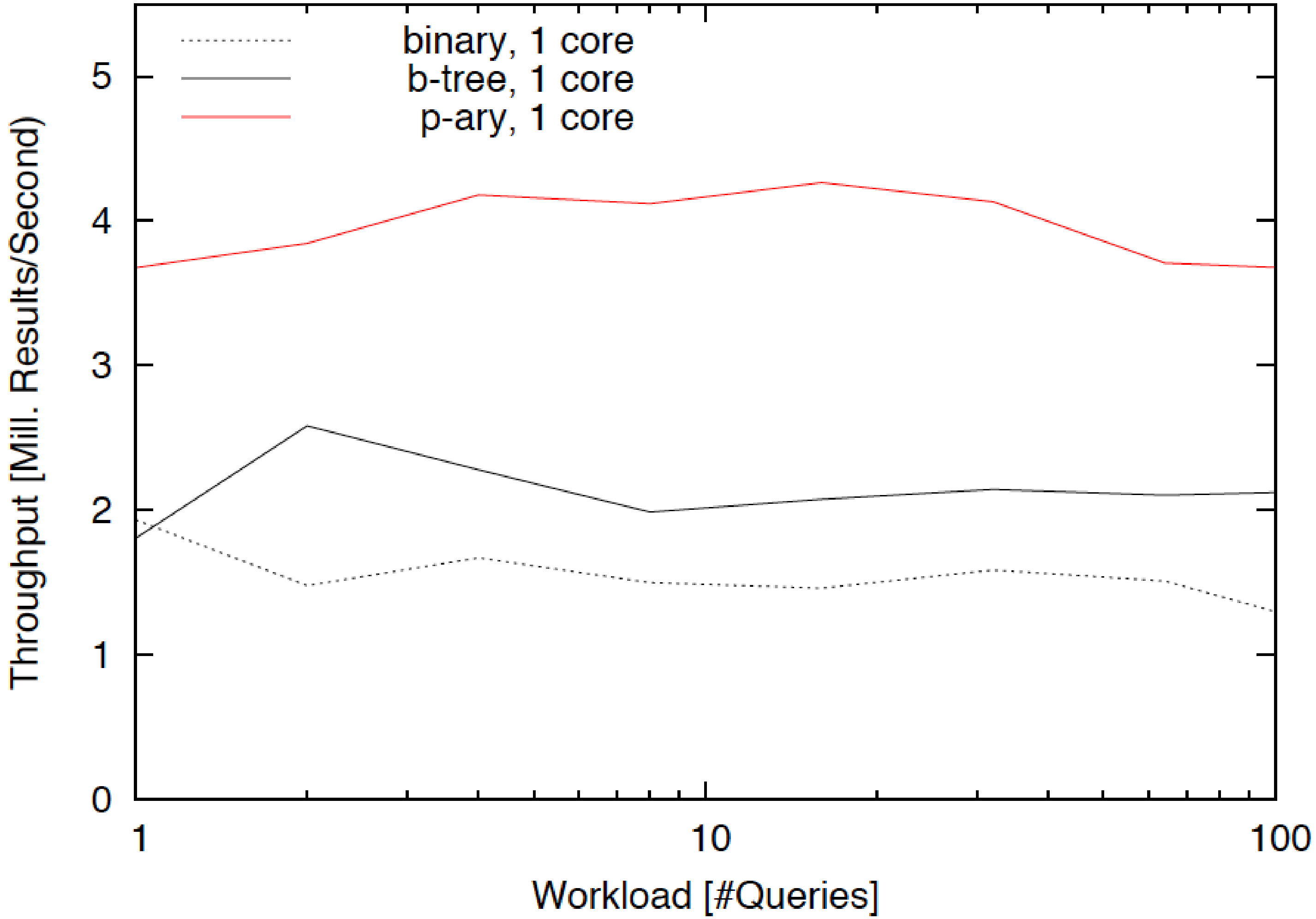
P-ary search - Implementation

- Performance depends on data structure
  - B-trees group pivot elements



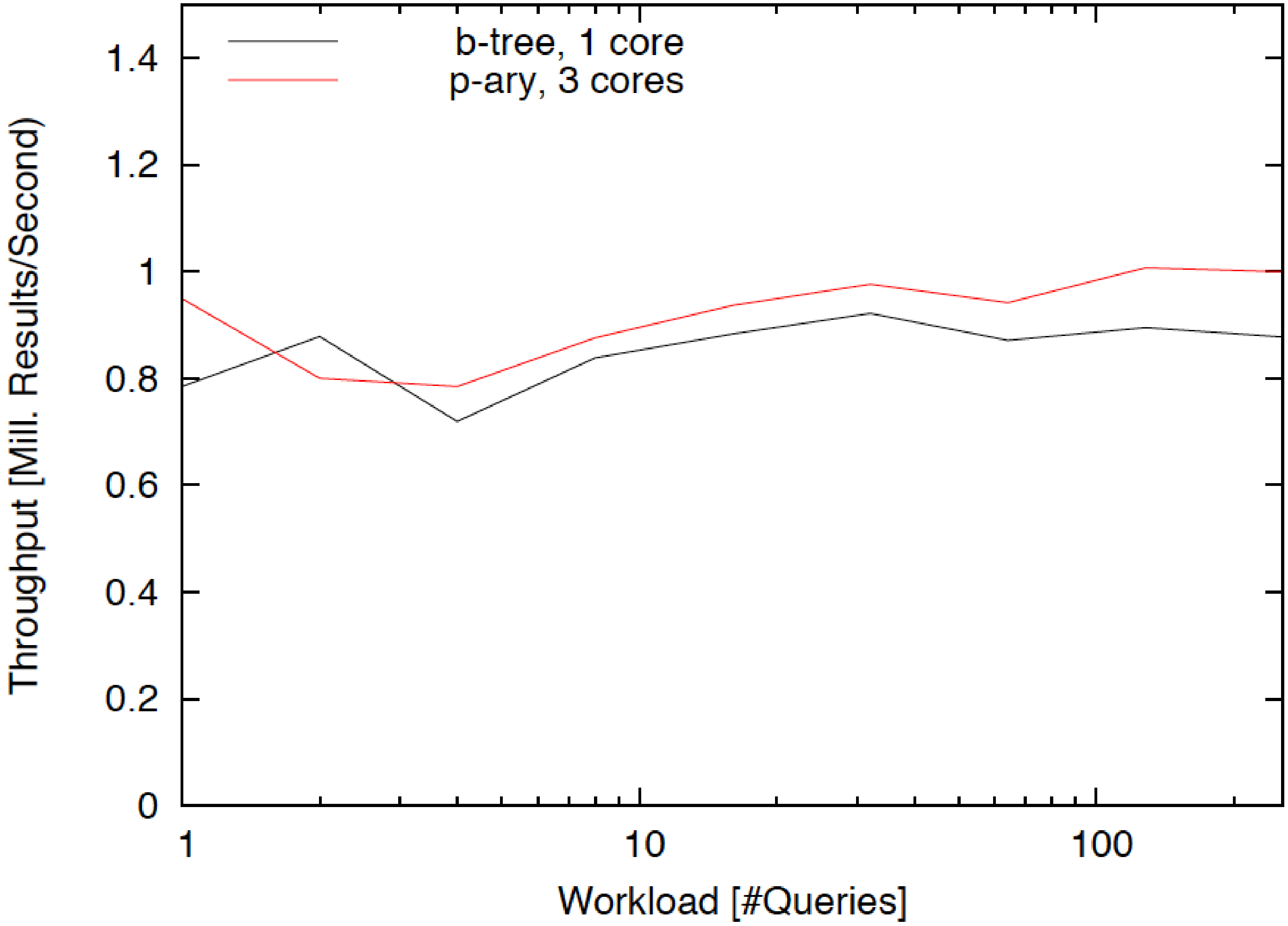
  - Linear memory accesses are fast
  - Nodes can also be mapped to
    - Cache Lines (CSB+ trees)
    - Vectors (SSE)

P-ary Search (SSE) vs. conventional algorithms



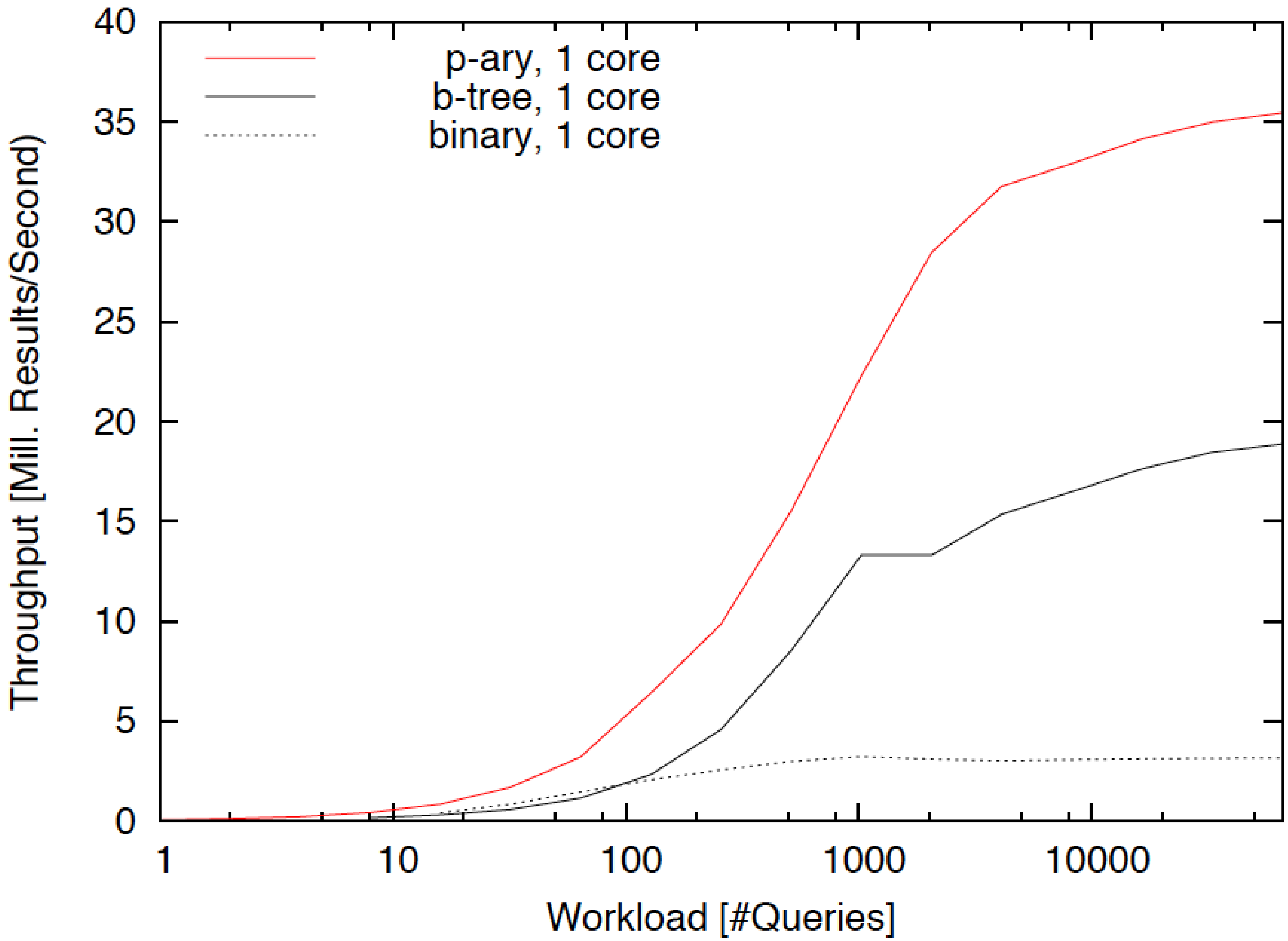
Searching a 512MB index with 134mill. 4-byte integer entries. Index stored as 4-wide (16-wide) B-tree. Results for a Core i7 2.66GHz, DDR3 1666.

P-ary Search (multi-core) vs. traditional (multi)-threading



Searching a 512MB index with 134mill. 4-byte integer entries. Index stored as 48-wide B-tree. Results for a Core i7 2.66GHz, DDR3 1.6 GHz

P-ary Search implemented on a GPU



Searching a 512MB index with 134mill. 4-byte integer entries. Index stored as 32-wide B-tree. Results for a nVidia GTX 285 1.5GHz, GDDR3 1.2GHZ

Predictable memory performance

- Measure latency of memory access using “rdtsc”
  - random accesses take ~350 cycles
  - Sequential accesses are hard to measure
    - In a sequence they take ~2 cycles on average
    - Intel optimization reference manual states 4 cycle latency for L1
- Applying these results to our search problem we get:

Algorithm	# memory accesses linear	random	theoretical wcet [#cycles]	measured wcet [# cycles]	Estimation error [%]
binary search	3	24	8412	8637	-2.61
csb	105	7	2870	2877	-0.24
p-ary (SSE)		18	6300	6593	-4.44

- The average case is much faster
  - Not all search keys are found within the last iteration
  - Multiple queries in sequence will result in Cache hits

Conclusions

- Memory performance can differ by 2 orders of magnitude dependent on:
  - access pattern: random/sequential, read/write
  - word size
  - concurrency(growing importance)
- Taking memory characteristics into account
  - Improves performance
    - p-ary search (concurrency, word size)

      works across architectures and data structures
    - strcmp (word size)
  - Allows to predict performance of memory bound apps
    - based on their memory access pattern
    - within +/- 5% of the worst case execution time

Future Work

- Evaluate p-ary search with
  - Wider vectors
  - More cores

- Manage system performance for memory bound applications (databases), i.e. schedule queries
  - Based on resource requirements (using available metadata)
  - With the “right” level of parallelism for a job

- Graduate soon ;-)